Kubernetes! Kubernetes! Kubernetes!

Running stateful applications on Kubernetes; an adventure into deploying production Elasticsearch cluster on Kubernetes

Ishwor Gurung ishwor@campaignmonitor.com
Adeep Rego adeep@campaignmonitor.com

Kubernetes Concepts

- Kubernetes is a system for managing **clusters of containers**, including **orchestration** and **scheduling**
- Pods are the basic **deployable units** in a cluster. Pods have one or more tightly coupled docker containers
- Services define **abstractions** across a logical set of Pods and a **policy** to access them
- StatefulSet, DaemonSet and Deployment ensure that Pods are running at any given time with varying levels of guaranties and properties
- Namespaces provide **virtual clusters**

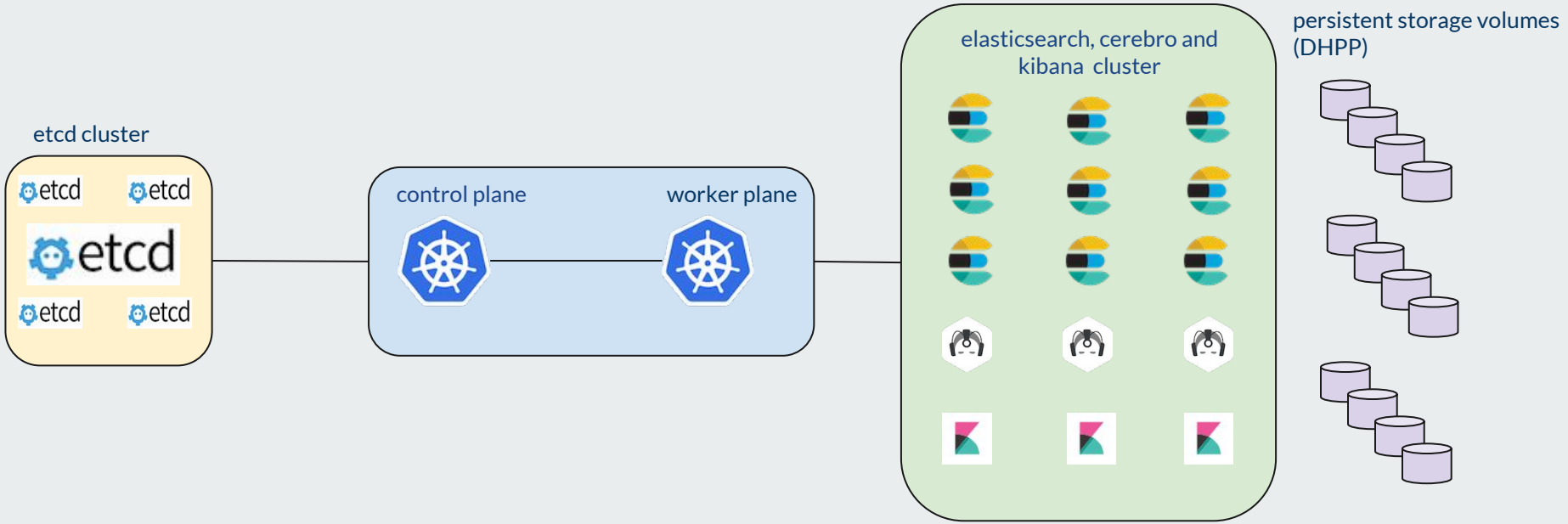
More Kubernetes Concepts

- Kubernetes objects are specified using **YAML** or **JSON**.
- The specifications of objects are submitted to Kubernetes API server running on controller nodes which **validate** it and transition the state of those managed resources to the **desired state** as specified in the spec.

Elasticsearch deployment at scale: Issues and Challenges

- How do we solve persistent storage problem?
- Can Stateful sets help us?
- Scale of data we are looking at: multi-terabytes
- How do we design it for HA and redundancy
  - Sharding strategy
  - Replication Factor
- Can we leverage Linux block-level replication for ES replication?

Kubernetes + Elasticsearch



CoreOS Tectonic

- Controllers and workers are provisioned using modified Terraform script
  - Controller nodes' are provisioned as a KVM guests using templating magic, CoreOS ignition with the guest configuration in XML
  - Controller nodes' template are modified to use static networking and override the default NTP servers
  - Worker nodes' template are modified to have
    - 4 x 10G bonded interfaces for max traffic throughput
    - Block device `/dev/sdb` (RAID 10) to automount on boot
- The etcd cluster is provisioned as a KVM guest using CoreOS ignition with the guest configuration in XML

Elasticsearch Network

- Layer3+4 bonded interfaces
- Four 10G interface between controllers, workers and etcd cluster
- 40G bonded inter-rack connectivity
- Services in elasticsearch are exposed via Kubernetes Services
- DNS is part of the Static Network Configuration in the templates

Software Infrastructure

- Kubernetes workers on bare metal CoreOS
  - Worker nodes run pods that are scheduled by the controllers
- Kubernetes control plane on KVM, dedicated etcd cluster on KVM
- ElasticSearch cluster run as Docker containers on the worker nodes
  - StatefulSets for the **ES data** nodes
    - High-availability achieved using Dynamic Host Path Provisioner Daemon Set (DHPP DS)
    - DHPP DS provides the persistent storage layer
    - Node affinity
  - ES Client, Master, Cerebro and Kibana pods are spun up as Replication Controller
    - No requirements to maintain state

Elasticsearch High-Availability

- In the event of a Kubernetes worker node failure, the pods are rescheduled to another node
- The data in elasticsearch pods are replicated using its built-in configurable replicator
  - In the event that a pod goes down, the data is persistent because of StatefulSet + DHPP DS
  - We trialled out block-level replication but did not choose to go down that path (we may come back to it in the future for other projects)

Elasticsearch Init Container

Container initialisation to increase `vm.max_map_count` sysctl value

```
annotations:
  pod.beta.kubernetes.io/init-containers : '[
    {
    "name": "sysctl",
      "image": "busybox",
      "imagePullPolicy": "IfNotPresent",
      "command": ["sysctl", "-w", "vm.max_map_count=262144"],
      "securityContext": {
        "privileged": true
      }
    }
  ]'
```

Elasticsearch volume health check

Define health checks for mounted data volume

```
        livenessProbe:
          exec:
            command: ["/bin/dd", "if=/dev/urandom", "of=/data/health", "bs=1M",
"count=1"]
          initialDelaySeconds: 5
          periodSeconds: 5
          timeoutSeconds: 1
          successThreshold: 1
          failureThreshold: 3
        readinessProbe:
          exec:
            command: ["/bin/dd", "if=/dev/urandom", "of=/data/health", "bs=1M",
"count=1"]
          initialDelaySeconds: 5
          periodSeconds: 5
          timeoutSeconds: 1
          successThreshold: 1
          failureThreshold: 3
```

Elasticsearch Persistent Storage

Define volume claim templates to be able to mount persistent volumes of type `hostpath-dynamic` (DHPP)

```
volumeClaimTemplates:
    - metadata:
        name: esVol
        annotations:
          volume.beta.kubernetes.io/storage-class:
hostpath-dynamic
```

Elasticsearch Configuration

Mount ES configuration

```
volumeMounts:
- name: esvol
  mountPath: /data
- name: elastic-config
  mountPath: /elasticsearch/config/elasticsearch.yml
  subPath: elasticsearch.yml
```

Elasticsearch Configmaps

elasticsearch.yml as Config Maps

```
data:
 elasticsearch.yml : |-
   cluster:
     name: ${CLUSTER_NAME}
     routing:
       allocation:
         awareness:
           attributes : node_name

   node:
     master: ${NODE_MASTER}
     data: ${NODE_DATA}
     name: ${NODE_NAME}
     ingest: ${NODE_INGEST}
     max_local_storage_nodes : ${MAX_LOCAL_STORAGE_NODES}
     attr:
       node_name : ${REAL_NODE_NAME}

   network.host : ${NETWORK_HOST}
   path:
     data: /data/data
     logs: /data/log

   bootstrap:
     memory_lock: true

   http:
     enabled: ${HTTP_ENABLE}
     compression: true
     cors:
       enabled: ${HTTP_CORS_ENABLE}
       allow-origin: ${HTTP_CORS_ALLOW_ORIGIN}

   discovery:
     zen:
       ping.unicast.hosts: ${DISCOVERY_SERVICE}
       minimum_master_nodes: ${NUMBER_OF_MASTERS}
kind: ConfigMap
metadata:
 name: elastic-config
```

Elasticsearch front-end

Demo of Cerebro

Learnings (1)

- Lot of sleepless nights
- Persistent storage
  - Random disk failures whilst trialing out few of the container storage layer
    - Tested many different container storage layer and found some very complex setup
    - One snake-oil solution was also found
- CoreOS specific
  - The auto-update strategy needs to be configured before go live or it might randomly decided to update the CoreOS kernel on the node(s)
  - Stay on a stable channel release unless there is a strong reason not to
  - There were few issues with random kernel panics
    - CoreOS kernel issues

Learnings (2)

- Provision etcd separately outside the kubernetes cluster
  - States are stored in etcd (chicken/egg problem)
- Along the way, we became beta testers for CoreOS
  - Found many bugs in their code
  - Consider the time, effort and resource allocations needed to test bleeding edge stack

Q&A / Comments

Thanks!